

History-Register Automata

Radu Grigore and Nikos Tzevelekos

Queen Mary, University of London

Abstract. Programs with dynamic allocation are able to create and use an unbounded number of fresh resources, such as references, objects, files, etc. We propose History-Register Automata (HRA), a new automata-theoretic formalism for modelling and verifying such programs. HRAs extend the expressiveness of previous approaches and bring us to the limits of decidability for reachability checks. The distinctive feature of our machines is their use of unbounded memory sets (histories) where input symbols can be selectively stored and compared with symbols to follow. In addition, stored symbols can be consumed or deleted by reset. We show that the combination of consumption and reset capabilities renders the automata powerful enough to imitate counter machines, and in particular reset Petri nets, and yields closure under all regular operations apart from complementation. We moreover examine weaker notions of HRAs which strike different balances between expressiveness and effectiveness.

1 Introduction

Program analysis faces substantial challenges due to its aim to devise finitary methods and machines which are called to operate on potentially infinite program computations. A specific such challenge stems from dynamic generative behaviours such as, for example, object or thread creation in Java, or reference creation in ML. A program engaging in such behaviours is expected to generate a possibly unbounded amount of distinct, i.e. *fresh*, resources, each of which is assigned a unique identifier, a *name*. Hence, any machine designed for analysing such programs is expected to operate on an infinite alphabet of names. The latter need has brought about introducing automata over infinite alphabets in program analysis, starting from prototypical machines for mobile calculi [21] and variable programs [17], and recently developing towards automata for verification tasks such as equivalence checks of ML programs [22,23], context-bounded analysis of concurrent programs [7,3] and runtime program monitoring [13].

The literature on automata over infinite alphabets is rich in formalisms each based on a different approach for tackling the infiniteness of the alphabet in a finitary manner (see e.g. [28] for an overview). A particularly intuitive such model is that of *Register Automata (RA)* [17,24], which are machines built around the concept of an ordinary finite-state automaton attached with a fixed finite amount of registers. The automaton can store in its registers names coming from the input, and make control decisions by comparing new input names with those already stored. Thus, by talking about addresses of its memory registers

rather than actual names, a so finitely-described automaton can tackle the infinite alphabet of names. Driven by program analysis considerations, register automata have been recently extended with feature of name-freshness recognition [30], that is, the capability of the automaton to accept specific inputs just if they are *fresh* – they have not appeared before during computation. Those automata, called *Fresh-Register Automata (FRA)*, can account for languages like the following,

$$\mathcal{L}_0 = \{a_1 \cdots a_n \in \mathcal{N}^* \mid \forall i \neq j. a_i \neq a_j\}$$

which captures the output of a fresh-name generator (\mathcal{N} is a countably infinite set of names). FRAs are expressive enough to model, for example, finitary fragments of languages like the π -calculus [30] or ML [22].

The freshness oracle of FRAs administers the automata to have some restricted access to the full history of the computation. In this work we further capitalise on the use of histories by effectively upgrading them to the status of registers. That is, in addition to registers, we equip our automata with a fixed number of unbounded sets of names (*histories*) where input names can be stored and compared with names to follow. As histories are internally unordered, the kind of name comparison we allow for is name belonging (*does the input name belong to the i -th history?*). Moreover, names can be selected and removed from histories, and individual histories can be reset altogether. We call the resulting machines *History-Register Automata (HRA)*.

The above generalisations greatly strengthen the expressive power of our machines. For example, different input names may be stored in distinct histories and checked for different properties. Moreover, individual names can be removed from histories, thus allowing us to express *consumption* of resources. More specifically, we identify three distinctive features of HRAs:

- (a) The capability to reset histories, which captures languages like

$$\mathcal{L}_1 = \{a_0 w_1 \cdots a_0 w_n \in \mathcal{N}^* \mid \forall i. w_i \in \mathcal{N}^* \wedge a_0 w_i \in \mathcal{L}_0\}$$

for some fixed name a_0 .

- (b) The use of multiple histories, which allows one to express

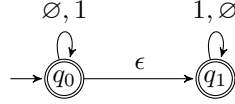
$$\mathcal{L}_2 = \{a_1 a'_1 \cdots a_n a'_n \in \mathcal{N}^* \mid a_1 \cdots a_n, a'_1 \cdots a'_n \in \mathcal{L}_0\}.$$

- (c) The capability to select and remove individual names from histories, which allows us to express

$$\mathcal{L}_3 = \{a_1 \cdots a_n a'_1 \cdots a'_{n'} \in \mathcal{N}^* \mid a_1 \cdots a_n, a'_1 \cdots a'_{n'} \in \mathcal{L}_0 \wedge \forall i. \exists j. a'_i = a_j\}.$$

The above examples, neither of which are FRA-recognisable, are used here to demonstrate the limitations of FRAs: although the latter are sufficiently powerful for expressing the semantics of programs with generative effects, they fall short of providing a satisfactorily rich verification toolkit for them. We cannot (a) Kleene-close FRAs (actually, not even concatenate them), (b) interleave them nor (c) use them to express non-freshness or consumption of names.

Let us further expand on point (c). The language \mathcal{L}_3 captures a paradigmatic scenario of a name generator followed by a name consumer: each consumed name a'_i must have been created first (non-freshness), and no name can be consumed twice. The language is decided by the following automaton with one history.



The automaton starts from state q_0 with empty history and, for each input name a , if a does not appear in the history, it accepts a , stores it in the history and goes back to q_0 .¹ Alternatively, the automaton can make an ϵ -transition to consumption mode, i.e. to state q_1 . There, for each input name a , if a already appears in the history, the automaton accepts a , removes it from the history and goes back to q_1 .² Thus, at q_0 the automaton acts as a name generator and at q_1 as a name consumer.

Apart from the gains in expressive power, the passage to HRAs yields a more well-rounded automata-theoretic formalism for generative behaviours as these machines enjoy closure under all regular operations apart from complementation (union, intersection, concatenation, Kleene star). On the other hand, the combination of the aforementioned features (a-c) of HRAs enable us to use histories as counters and simulate counter machines. We therefore obtain non-primitive recursive bounds for checking language emptiness. Given that language containment and universality are undecidable already for register automata [24], HRAs are fairly close to the decidability boundary for properties of languages over infinite alphabets. Nonetheless, starting from HRAs and weakening them in each of the first two enumerated factors (a,b) we obtain automata models which are still highly expressive but computationally more tractable. Overall, the expressiveness hierarchy of the machines we examine is depicted in Figure 1 (weakening in (a) and (b) respectively occurs in the second column of the figure).

Motivation and related work

The motivation behind this work stems from semantics and verification. In semantics, the use of names to model resource generation originates in the work of Pitts and Stark on the ν -calculus [25] and Stark's PhD thesis [29]. Names have subsequently been incorporated in the semantics literature (see, for example [15,4,1,18]), especially after the advent of *Nominal Sets* [12] which provided formal foundations for doing mathematics with names. Recent work in game semantics in particular has produced algorithmic representations of game models

¹ Thus, the “ \emptyset ” in the loop label means that a does not appear in any history, and the “1” that, once accepted, a should be added to history number 1 (the only history of the automaton in this case).

² Thus, the “1” in this transition label means that a already appears in history 1, and the “ \emptyset ” that, once accepted, a should appear in no history.

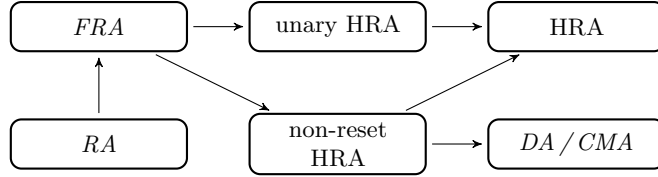


Fig. 1. Expressiveness of history-register automata compared to previous models (in italics). The inclusion $\mathcal{M} \rightarrow \mathcal{M}'$ means that for each $\mathcal{A} \in \mathcal{M}$ we can effectively construct an $\mathcal{A}' \in \mathcal{M}'$ accepting the same language as \mathcal{A} . All inclusions are strict.

using extensions of fresh-register automata [22,23], thus achieving automated equivalence checks for fragments of ML. In a parallel development, a research stream on automated analysis of dynamic concurrent programs has developed essentially the same formalisms, this time stemming from basic operational semantics [7,3]. This confluence of different methodologies is exciting and encourages the development of stronger automata for a wider range of verification tasks, and just such an automaton we propose herein.

Although our work is driven by program analysis, the closest existing automata models to ours come from XML database theory and model checking. Research in the latter area has made great strides in the last years on automata over infinite alphabets and related logics (e.g. see [28] for an overview from 2006). As we show in this paper, history-register automata fit very well inside the big picture of automata over infinite alphabets (cf. Figure 1) and in fact can be seen as a variant of *Data Automata (DA)* [6] or, equivalently *Class Memory Automata (CMA)* [5]. This fit leaves space for transfer of technologies and, more specifically, of the associated logics of data automata.

Overview

The next section introduces HRAs and looks into examples and some first properties. In Section 3 we examine regular closure properties of HRAs and in Section 4 we prove decidability for emptiness. In Section 5 we introduce weaker models and in Section 6 we connect HRAs to existing automata formalisms. We conclude by discussing future directions which emanate from this work.

2 Definitions and first properties

We start by fixing some notation. Let \mathcal{N} be a countably infinite alphabet of *names*, which we range over by a, b, c , etc. For any pair of natural numbers $i \leq j$, we write $[i, j]$ for the set $\{i, i+1, \dots, j\}$, and for each i we let $[i]$ be the set $\{1, \dots, i\}$. For any set S , we write $|S|$ for the cardinality of S , $\mathcal{P}(S)$ for the powerset of S , $\mathcal{P}_{\text{fin}}(S)$ for the set of finite subsets of S , and $\mathcal{P}_{\neq \emptyset}(S)$ for the set of non-empty subsets of S . We write $\text{id} : S \rightarrow S$ for the identity function on S , and $\text{img}(f)$ for the image of $f : S \rightarrow T$.

We define automata which are equipped with a fixed number of **registers** and **histories** where they can store names. Each register is a memory cell where one name can be stored at a time; each history can hold an unbounded set of names. We use the term **place** to refer to both histories and registers.

Transitions are of two kinds: name-accepting transitions and reset transitions. Those of the former kind have labels of the form (X, X') , for sets of places X and X' ; and those of the latter carry labels with single sets of places X . A transition labelled (X, X') means:

- accept name a if it is contained precisely in places X , and
- update places in X and X' so that a be contained precisely in places X' after the transition (without touching other names).

By a being contained precisely in places X we mean that it appears in every place in X , and in no other place. In particular, the label (\emptyset, X') signifies accepting a fresh name (one which does not appear in any place) and inserting it in places X' . On the other hand, a transition labelled by X resets all the places in X , that is, updates each to be the empty set. Reset transitions do not accept names; they are ϵ -transitions from the outside. Note then that the label (X, \emptyset) has different semantics from the label X : the former stipulates that a name appearing precisely in X be accepted and then removed from X ; whereas the latter clears all the contents of the places in X , without accepting anything.

Formally, let us fix positive integers m and n which will stand for the default number of histories and registers respectively in the machines we define below. The set **Asn** of **assignments** and the set **Lab** of **labels** are:

$$\begin{aligned}\text{Asn} &= \{H : [m+n] \rightarrow \mathcal{P}_{\text{fin}}(\mathcal{N}) \mid \forall i > m. |H(i)| \leq 1\} \\ \text{Lab} &= \mathcal{P}([m+n])^2 \cup \mathcal{P}([m+n])\end{aligned}$$

For example, $\{(i, \emptyset) \mid i \in [m+n]\}$ is the empty assignment. We range over elements of **Asn** by H and variants, and over elements of **Lab** by ℓ and variants. Moreover, it will be handy to introduce the following notation for assignments. For any assignment H and any $a \in \mathcal{N}$, $S \subseteq \mathcal{N}$ and $X \subseteq [m+n]$:

- We set $H@X$ to be the set of names which *appear precisely* in places X in H , that is, $H@X = \bigcap_{i \in X} H(i) \setminus \bigcup_{i \notin X} H(i)$. In particular, $H@\emptyset = \mathcal{N} \setminus \bigcup_{i \in [m+n]} H(i)$ is the set of names which are fresh for H .
- $H[X \mapsto S]$ is the update H' of H so that all places in X are mapped to S , i.e. $H' = \{(i, H(i)) \mid i \notin X\} \cup \{(i, S) \mid i \in X\}$. For example, $H[X \mapsto \emptyset]$ resets all places in X .
- $H[a \text{ in } X]$ is the update of H which removes name a from all places and inserts it back in X , that is, for all i :

$$H[a \text{ in } X](i) = \begin{cases} H(i) \setminus \{a\} & i \notin X \\ H(i) \cup \{a\} & i \in X \cap [m] \\ \{a\} & i \in X \setminus [m] \end{cases}$$

Note above that operation $H[a \text{ in } X]$ acts differently in the case of histories ($i \leq m$) and registers ($i > m$) in X : in the former case, the name a is added to the history $H(i)$, while in the latter the register $H(i)$ is set to $\{a\}$ and its previous content is cleared.

We can now define our automata.

Definition 1. A **history-register automaton (HRA)** of type (m, n) is a tuple $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ where:

- Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ are the final ones,
- $H_0 \in \text{Asn}$ is the initial assignment, and
- $\delta \subseteq Q \times \text{Lab} \times Q$ is the transition relation.

For brevity, we shall call \mathcal{A} an (m, n) -HRA.

We write transitions in the forms $q \xrightarrow{X, X'} q'$ and $q \xrightarrow{X} q'$, for each kind of transition labels. In diagrams, we may unify different transitions with common source and target, for example $q \xrightarrow{X, X'} q'$ and $q \xrightarrow{Y, Y'} q'$ may be written $q \xrightarrow{X, X' / Y, Y'} q'$; moreover, we shall lighten notation and write i for the singleton $\{i\}$, and ij for $\{i, j\}$.

We already gave an overview of the semantics of HRAs. This is formally defined by means of configurations representing the current computation state of the automaton. A **configuration** of \mathcal{A} is a pair $(q, H) \in \hat{Q}$, where:

$$\hat{Q} = Q \times \text{Asn}$$

From the transition relation δ we obtain the configuration graph of \mathcal{A} as follows.

Definition 2. Let \mathcal{A} be an (m, n) -HRA as above. Its **configuration graph** $(\hat{Q}, \longrightarrow)$, where $\longrightarrow \subseteq \hat{Q} \times (\mathcal{N} \cup \{\epsilon\}) \times \hat{Q}$, is constructed by setting $(q, H) \xrightarrow{x} (q', H')$ just if one of the following conditions is satisfied.

- $x = a \in \mathcal{N}$ and there is $q \xrightarrow{X, X'} q' \in \delta$ such that $a \in H @ X$ and $H' = H[a \text{ in } X']$.
- $x = \epsilon$ and there is $q \xrightarrow{X} q' \in \delta$ such that $H' = H[X \mapsto \emptyset]$.

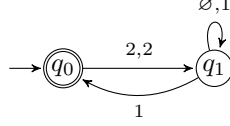
The language accepted by \mathcal{A} is:

$$\mathcal{L}(\mathcal{A}) = \{ w \in \mathcal{N}^* \mid (q_0, H_0) \xrightarrow{w} (q, H) \text{ and } q \in F \}$$

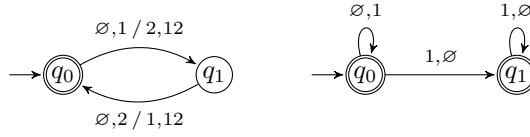
where $\xrightarrow{}^*$ is the reflexive transitive closure of \longrightarrow , that is, $\hat{q} \xrightarrow{x_1 \cdots x_n} \hat{q}'$ if $\hat{q} \xrightarrow{x_1} \cdots \xrightarrow{x_n} \hat{q}'$.

Note that we use ϵ both for the empty sequence and the empty transition so, in particular, when writing sequences of the form $x_1 \cdots x_n$ we may implicitly consume ϵ 's.

Example 3. The language \mathcal{L}_1 of the Introduction is recognised by the following $(1, 1)$ -HRA with initial assignment $\{(1, \emptyset), (2, a_0)\}$.



The automaton starts by accepting a_0 , leaving it in register 2, and moving to state q_1 . There, it loops accepting fresh names (appearing in no place) which it stores in history 1. From q_1 it goes back to q_0 by resetting its history. We can also see that the following HRAs, of type $(2, 0)$ and $(1, 0)$, accept the languages \mathcal{L}_2 and \mathcal{L}_3 respectively.



Both automata start with empty assignments.

As mentioned in the introductory section, HRAs build upon *(Fresh) Register Automata* [17,24,30]. The latter can be defined within the HRA framework as follows.³

Definition 4. A **Fresh-Register Automaton (FRA)** of n registers is a $(1, n)$ -HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ such that:

- $H_0(1) = \bigcup_i H_0(i)$ and, for all $(q, \ell, q') \in \delta$, there are X, X' such that $\ell = (X, X')$ and $1 \in X'$;
- for all $(q, \{1\}, X', q') \in \delta$, there is also $(q, \emptyset, X', q') \in \delta$.

A **Register Automaton (RA)** of n registers is a $(0, n)$ -HRA with no reset transitions.

Thus, in an FRA all the initial names must appear in its history, and the same holds for all the names the automaton accepts during computation ($1 \in X'$). As, in addition, no reset transitions are allowed, the history effectively contains all names of a run. On the other hand, the automaton cannot recognise *non-freshness*: if a name appearing only in the history is to be accepted at any point then a totally fresh name can be also be accepted in the same way. Now, from [30] we have the following.

Lemma 5. The languages $\mathcal{L}_1, \mathcal{L}_2$ and \mathcal{L}_3 are not FRA-recognisable.

Proof. \mathcal{L}_1 was explicitly examined in [30]. For \mathcal{L}_2 and \mathcal{L}_3 we use a similar argument as the one for showing that $\mathcal{L}_0 * \mathcal{L}_0$ is not FRA-recognisable [30]. \square

³ The definitions given in [17,24,30] are slightly different but they can routinely be shown equivalent.

Bisimulation Bisimulation equivalence, also called *bisimilarity*, is a useful tool for relating automata, even from different paradigms. It implies language equivalence and is generally easier to reason about than the latter. We will be using it avidly in the sequel.

Definition 6. Let $\mathcal{A}_i = \langle Q_i, q_{0i}, H_{0i}, \delta_i, F_i \rangle$ be (m, n) -HRAs, for $i = 1, 2$. A relation $R \subseteq \hat{Q}_1 \times \hat{Q}_2$ is called a *simulation* on \mathcal{A}_1 and \mathcal{A}_2 if, for all $(\hat{q}_1, \hat{q}_2) \in R$,

- if $\hat{q}_1 \xrightarrow{\epsilon} \hat{q}'_1$ and $\pi_1(\hat{q}'_1) \in F_1$ then $\hat{q}_2 \xrightarrow{\epsilon} \hat{q}'_2$ for some $\pi_1(\hat{q}'_2) \in F_2$, where π_1 the first projection function;
- if $\hat{q}_1 \xrightarrow{\epsilon} \cdot \xrightarrow{a} \hat{q}'_1$ then $\hat{q}_2 \xrightarrow{\epsilon} \cdot \xrightarrow{a} \hat{q}'_2$ for some $(\hat{q}'_1, \hat{q}'_2) \in R$.

R is called a **bisimulation** if both R and R^{-1} are simulations. We say that \mathcal{A}_1 and \mathcal{A}_2 are **bisimilar**, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there is a bisimulation R such that $((q_{01}, H_{01}), (q_{02}, H_{02})) \in R$.

The following is a standard result.

Lemma 7. If $\mathcal{A}_1 \sim \mathcal{A}_2$ then $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.

As a first taste of HRA reasoning, we demonstrate a technique for simulating registers by histories in HRAs. The idea is to represent a register by a history whose size is always kept at most 1. There are, however, some technicalities. To ensure that histories are effectively kept in size ≤ 1 they must be cleared before inserting names. This in turn complicates the conditions dictating when a transition can be taken as such conditions may depend on the deleted names.

Proposition 8. Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ be an (m, n) -HRA. There is an $(m+2n, 0)$ -HRA \mathcal{A}' such that $\mathcal{A} \sim \mathcal{A}'$.

Proof. To each of the n registers of \mathcal{A} we assign a pair of histories in \mathcal{A}' . The reason we need two copies of each register is so that, for each transition with label (X, X') , we use one copy for the name comparisons needed for the X part of the label (old copy), and the other copy for the assignments dictated by X' (new copy). Note that assigning names in the same copy would mean that our histories would grow to sizes greater than 1. After the assignment, the old copies of X and X' are garbage. The correspondence is completed by pre-composing each such transition with a reset of all garbage.

Formally, $\mathcal{A}' = \langle Q', q'_0, H'_0, \delta', F' \rangle$ where elements of Q' are of the form (q, f) with $q \in Q$ and $f : [m+1, m+n] \rightarrow [m+1, m+2n]$ is such that, for each i , $f(i) \in \{i, i+n\}$.⁴ The role of f is to record which copy of the two registers is currently used. We set \bar{f} to be the complement of f , that is, the function defined by $\bar{f}(i) = n+2i - f(i)$. We write $f^\dagger : [m+n] \rightarrow [m+2n]$ for the function given by $f^\dagger(i) = i$ if $i \in [m]$ and $f(i)$ otherwise. Moreover, $q'_0 = (q_0, \text{id})$, $F' = \{(q, f) \mid q \in F\}$, and H'_0 is H_0 so extended that $H'_0(i) = \emptyset$ for all $i > m+n$. Finally, we include in δ' precisely the following transitions.

⁴ Since each (X, X') -labelled transition in δ decomposes into two transitions in δ' , we also need dummy states in between but we prefer to gloss over this easy point for economy.

- For each $q \xrightarrow{X, X'} q' \in \delta$, add $(q, f) \xrightarrow{Y} \cdot \xrightarrow{f^\dagger(X), \bar{f}^\dagger(X')} (q', f')$ where $Y = [m+1, m+2n] \setminus \text{img}(f)$, and f' is given by: $f'(i) = \bar{f}(i)$ if $i \in X \cup X'$ and $f'(i) = f(i)$ otherwise.
- For each $q \xrightarrow{X} q' \in \delta$, add $(q, f) \xrightarrow{f^\dagger(X)} (q', f)$.

Setting $R = \{((q, H), (q, f, H')) \mid H = H' \circ f^\dagger\}$, we have that R witnesses bisimilarity. \square

The above reduction makes substantial use of reset transitions and of transitions which move names between histories. As shown in [5], it is possible to express register behaviour with histories and without resets, using a so called *colouring technique*. The latter is demonstrated in Example 25 of Section 6 but we do not have a general concrete reduction for HRAs. More In generally, though, the technique obscures the intuition of registers and produces automata which need close examination even for simple languages like the one which contains all words $a_1 \cdots a_n$ such that $a_i \neq a_{i+1}$ for all i (see Example 25). As, in addition, it is not applicable to the weaker unary HRAs we examine in Section 5, we preferred to explicitly include registers in HRAs. Another design choice regards the use of sets of places in transitions instead e.g. of single places. Although the latter description would lead to an equivalent and probably conciser formalism, it would be inconvenient for combining HRAs e.g. in order to produce the intersection of their accepted languages. In fact, our formulation follows *M-automata* [17], an equivalent presentation of RAs susceptible to closure constructions.

Determinism We close our presentation here by describing the deterministic class of HRAs. We defined HRAs in such a way that, at any given configuration (q, H) and for any input symbol a , there is at most one set of places X that can match a , i.e. such that $a \in H @ X$. As a result, the notion of determinism in HRAs can be ensured by purely syntactic means. Below we write $q \xrightarrow{X} q' \in \delta$ if there is a sequence of transitions $q \xrightarrow{X_1} \cdots \xrightarrow{X_n} q'$ in δ such that $X = \bigcup_{i=1}^n X_i$. In particular, $q \xrightarrow{\emptyset} q \in \delta$.

Definition 9. Let \mathcal{A} be an HRA. We say that \mathcal{A} is **deterministic** if, for any reachable configuration \hat{q} and any name a , if $\hat{q} \xrightarrow{\epsilon} \cdot \xrightarrow{a} \hat{q}_1$ and $\hat{q} \xrightarrow{\epsilon} \cdot \xrightarrow{a} \hat{q}_2$ then $\hat{q}_1 = \hat{q}_2$.

We say that \mathcal{A} is **strongly deterministic** if $q \xrightarrow{Y_1} \cdot \xrightarrow{X \setminus Y_1, X_1} q_1 \in \delta$ and $q \xrightarrow{Y_2} \cdot \xrightarrow{X \setminus Y_2, X_2} q_2 \in \delta$ imply $q_1 = q_2$, $Y_1 = Y_2$ and $X_1 = X_2$.

Lemma 10. If \mathcal{A} is strongly deterministic then it is deterministic.

3 Closure properties

History-register automata enjoy good closure properties with respect to regular language operations. In particular, they are closed under union, intersection, concatenation and Kleene star, but not closed under complementation.

In fact, the design of HRAs is such that the automata for union and intersection come almost for free through a straightforward product construction which is essentially an ordinary product for finite-state automata, modulo reindexing of places to account for duplicate labels (cf. [17]). The constructions for Kleene star and complementation are slightly more involved. We shall need the following technical gadget, which is fully presented in Appendix B. Given an (m, n) -HRA \mathcal{A} and a sequence w of k distinct names, we construct a bisimilar $(m, n+k)$ -HRA, denoted $\mathcal{A} \text{ fix } w$, in which the names of w appear exclusively in the additional k registers, which, moreover, remain unchanged during computation. The construction will allow us, for instance, to create feedback loops in automata ensuring that after each feedback transition the same initial configuration occurs.

Lemma 11. *Let \mathcal{A} be an (m, n) -HRA with initial assignment H_0 and $w = a_1 \cdots a_k$ a sequence of distinct names. We can effectively construct an $(m, n+k)$ -HRA $\mathcal{A} \text{ fix } w$ with initial assignment H'_0 such that:*

- $H'_0(m+n+i) = a_i$ for all $i \in [k]$, and $H'_0(i) = H_0(i) \setminus \{a_1, \dots, a_k\}$ for all $i \in [m+n]$;
- for all reachable configurations (q, H) of $\mathcal{A} \text{ fix } w$ and all $i > m+n$, $H(i) = H'_0(i)$.

We can now show the following.

Proposition 12. *Languages recognised by HRAs are closed under union, intersection, concatenation and Kleene star.*

Proof. We show concatenation and Kleene star only. For the former, consider HRAs $\mathcal{A}_i = \langle Q_i, q_{0i}, H_{0i}, \delta_i, F_i \rangle$, $i = 1, 2$, and assume wlog that they have common type (m, n) . Let w be an enlistment of all names in H_{02} and construct $\mathcal{A}'_i = \mathcal{A}_i \text{ fix } w$, for $i = 1, 2$. Then, $\mathcal{L}(\mathcal{A}_1) * \mathcal{L}(\mathcal{A}_2)$ is the language recognised by connecting \mathcal{A}'_1 and \mathcal{A}'_2 serially, that is, the automaton obtained by connecting each final state of \mathcal{A}'_1 to the initial state of \mathcal{A}'_2 with a transition labelled $[m+n]$, and with initial/final states those of $\mathcal{A}'_1/\mathcal{A}'_2$ respectively.

Finally, given an (m, n) -HRA \mathcal{A} and an enlistment w of its initial names, we construct an automaton \mathcal{A}' by connecting the final states of $\mathcal{A} \text{ fix } w$ to its initial state with a transition labelled $[m+n]$. We can see that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})^*$. \square

In the next section we shall see that, while universality is undecidable for HRAs, their emptiness problem can be decided by reduction to coverability for transfer-reset vector addition systems. In combination, these results imply that HRAs cannot be effectively complemented. In fact, the following holds.

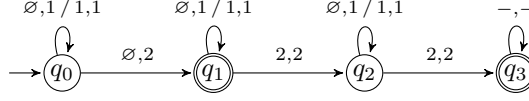
Lemma 13. *HRAs are not closed under complementation.*

The proof is in the following example, adapted from [20].

Example 14. Consider the following language and its complement.

$$\begin{aligned} \mathcal{L}_4 &= \{w \in \mathcal{N}^* \mid w \neq \epsilon \wedge \text{not all names in } w \text{ occur exactly twice in it} \} \\ \overline{\mathcal{L}_4} &= \{w \in \mathcal{N}^* \mid \text{all names in } w \text{ occur exactly twice in it} \} \end{aligned}$$

\mathcal{L}_4 is accepted by the following $(2, 0)$ -HRA, where “ $-$ ” can be any of $\emptyset, \{1\}, \{2\}$.



The automaton non-deterministically selects an input name which either appears only once in the input or at least three times.

However, $\overline{\mathcal{L}_4}$ is not HRA-recognisable. For suppose it were recognisable (wlog) by an $(m, 0)$ -HRA \mathcal{A} with k states. Then, \mathcal{A} would accept the word

$$w = a_1 \cdots a_k a_1 \cdots a_k$$

where all a_i 's are distinct and do not appear in the initial assignment of \mathcal{A} . Let $p = p_1 p_2$ be the path in \mathcal{A} through which w is accepted, with each p_i corresponding to one of the two halves of w . Since all a_i s are fresh for \mathcal{A} , the non-reset transitions of p_1 must carry labels of the form (\emptyset, X) , for some sets X . Let q be a state appearing twice in p_1 , say $p_1 = p_{11}(q)p_{12}(q)p_{13}$. Consider now the path $p' = p'_1 p_2$ where p'_1 is the extension of p_1 which repeats p_{12} , that is, $p'_1 = p_{11}(q)p_{12}(q)p_{12}(q)p_{13}$. We claim that p' is an accepting path in \mathcal{A} . Indeed, by our previous observation on the labels of p_1 , the path p'_1 does not block, i.e. it cannot reach a transition $q_1 \xrightarrow{X,Y} q_2$, with $X \neq \emptyset$, in some configuration (q_1, H_1) such that $H_1 @ X = \emptyset$. We need to show that p_2 does not block either (in p'). Let us denote (q, H_1) and (q, H_2) the configurations in each of the two visits of q in the run of p on w ; and let us write (q, H_3) for the third visit in the run of p'_1 , given that for the other two visits we assume the same configurations as in p . Now observe that, for each non-empty $X \subseteq [m]$, repeating p_{12} cannot reduce the number of names appearing precisely in X , therefore $|H_2 @ X| \leq |H_3 @ X|$. The latter implies that, since p does not block, p' does not block either. Now observe that any word accepted by w' is not in $\overline{\mathcal{L}_4}$, as p'_1 accepts more than k distinct names, a contradiction.

4 Emptiness and Universality

We now turn to the question of checking emptiness. The use of unbounded histories effectively renders our machines into counter automata: where a counter automaton would increase (or decrease) a counter, an HRA would add (remove) a name from one of its histories, or set of histories. Nonetheless, HRAs cannot decide their histories for emptiness, which leaves space for decidability.⁵ The capability for resetting histories leads us to consider Transfer-Reset Vector Addition Systems [8,2] (i.e. Petri nets with reset and transfer arcs) as equivalent formalisms for checking emptiness.

A **Transfer-Reset Vector Addition System (TR-VASS)** of m dimensions will be a tuple $\mathcal{A} = \langle Q, \delta \rangle$ where Q a set of states and

$$\delta \subseteq Q \times (\{-1, 0, 1\}^m \cup [m]^2 \cup [m]) \times Q$$

⁵ Recall that 2-counter machines with increase, decrease and check for zero operations are Turing complete.

a transition relation. Each dimension of \mathcal{A} corresponds to an unbounded counter. Thus, a transition of \mathcal{A} can either update its counters by addition of a vector $\vec{v} \in \{-1, 0, 1\}^m$, or transfer the value of one counter to another, or reset some counter.

Formally, a configuration of \mathcal{A} is a pair $(q, \vec{v}) \in Q \times \mathbb{N}^m$ consisting of a state and a vector of values stored in the counters. The configuration graph of \mathcal{A} is constructed by including an edge $(q, \vec{v}) \rightarrow (q', \vec{v}')$ if:

- there is some $(q, \vec{v}'', q') \in \delta$ such that $\vec{v}' = \vec{v} + \vec{v}''$, or
- there is $(q, i, j, q') \in \delta$ such that $\vec{v}' = (\vec{v}[j \mapsto v_i + v_j])[i \mapsto 0]$,
- or there is some $(q, i, q') \in \delta$ such that $\vec{v}' = \vec{v}[i \mapsto 0]$;

where we write v_i for the i th dimension of \vec{v} , and $\vec{v}[i \mapsto v']$ for the update of \vec{v} where the i -th counter is set to v' . An **R-VASS** is a TR-VASS without transfer transitions.

The **control-state reachability** problem for TR-VASSs is defined as follows. Given a TR-VASS \mathcal{A} of m dimensions, a configuration (q_0, \vec{v}_0) and a state q , is there some $\vec{v} \in \mathbb{N}^m$ such that $(q_0, \vec{v}_0) \twoheadrightarrow (q, \vec{v})$? In such a case, we write $(\mathcal{A}, q_0, \vec{v}_0, q) \in \text{Reach}$.

Fact 15 ([9,27,10]) *Control-state reachability for TR-VASSs and R-VASSs is decidable and has non-primitive recursive complexity.*

We next reduce HRA emptiness to TR-VASS control-state reachability. By Proposition 8, we can consider HRAs without registers. Below we write $\vec{0}$ for the vector $0 \cdots 0$ and δ_i for $\vec{0}[i \mapsto 1]$.

Proposition 16. *Emptiness is decidable for HRAs.*

Proof. For each $(m, 0)$ -HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$, we construct a TR-VASS \mathcal{A}' with 2^m dimensions: one dimension \tilde{X} for each $X \subseteq [m]$. The dimension $\tilde{\emptyset}$ will be simply garbage collecting. We assign to each state of \mathcal{A} a corresponding state in \mathcal{A}' . Moreover,

- we map each $q \xrightarrow{X, X'} q'$ to a pair $q \xrightarrow{\vec{v}_X} \cdot \xrightarrow{-\vec{v}_{X'}} q'$, where $\vec{v}_Y = \delta_{\tilde{Y}}$ if $Y \neq \emptyset$ and $\vec{0}$ otherwise ($Y = X, X'$);
- we map each transition $q \xrightarrow{X} q'$ of \mathcal{A} to a sequence of transitions $q \xrightarrow{\tilde{X}_1, \tilde{Y}_1} \cdots \xrightarrow{\tilde{X}_i, \tilde{Y}_i} q'$ in \mathcal{A}' , where X_1, \dots, X_i an enumeration of all X_i such that $X_i \cap X \neq \emptyset$, and $Y_i = X_i \setminus X$;

We also add in \mathcal{A}' a state q_F and transitions $q \xrightarrow{\vec{0}} q_F$ for each $q \in F$, and set $\vec{v}_0 = \{(\tilde{X}, |H_0 @ X|) \mid \emptyset \neq X \subseteq [m]\} \cup \{(\tilde{\emptyset}, \emptyset)\}$.

\mathcal{A}' simulates the behaviour of \mathcal{A} so that, whenever \mathcal{A} is at a configuration (q, H) , \mathcal{A}' is at a configuration (q, \vec{v}) such that, for all non-empty $X \subseteq [m]$, $|H @ X| = v_{\tilde{X}}$. The transitions above capture exactly that. At a transition label (X, X') , a name appearing precisely in X is moved precisely to X' . On the other hand, the effect

of a reset label X is more complicated: all names appearing precisely in some set of places X_i such that $X_i \cap X \neq \emptyset$ will see themselves transferred precisely to $X_i \setminus X$.

We have that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there is an accepting run of \mathcal{A} , that is, a run from the initial to some final configuration which does not block, i.e. it does not reach a transition $q \xrightarrow{X,Y} q'$ in some configuration (q, H) such that $|H @ X| = 0$. Equivalently, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $(\mathcal{A}', q_0, \vec{v}_0, q_F) \in \text{Reach}$. Now we use Fact 15. \square

Doing the opposite reduction we can show that emptiness of even strongly deterministic HRAs is non-primitive recursive. In this direction, each R-VASS of m dimensions is simulated by an $(m, 0)$ -HRA so that the value of each counter i of the former is the same as the number of names appearing precisely in history i of the latter.

Proposition 17. *Checking emptiness of strongly deterministic HRAs is non-primitive recursive.*

Proof. Let \mathcal{A} be an R-VASS of m dimensions such that, for each transition $q \xrightarrow{\vec{v}} q'$, the number of non-zero dimensions of \vec{v} is 1. Moreover, suppose \mathcal{A} is deterministic in the following sense. For each state q and transitions $q \xrightarrow{t_1} q_1$ and $q \xrightarrow{t_2} q_2$,

- if $t_1 = \delta_i$, some i , then $t_1 = t_2$, and
- if $t_1 = t_2$ then $q_1 = q_2$.

By [27], control-state reachability for such R-VASSs is non-primitive recursive.⁶ Let $(\mathcal{A}, q_0, \vec{v}_0, q_F)$ be such an instance. We construct a strongly deterministic $(m, 0)$ -HRA \mathcal{A}' where each counter i of \mathcal{A} corresponds to the i -th history of \mathcal{A}' . For each i, j we write $i \oplus j$ for $i + j \bmod m$, and $i \ominus j$ for $i - j \bmod m$. For each dimension i with $v_{0i} = n$, we pick names a^i, a_1^i, \dots, a_n^i , all pairwise distinct. The names a_1^i, \dots, a_n^i will be used for representing the value n . On the other hand, each a^i will be used in reset transitions in order to ensure determinacy. We write \bar{i} for the set $\{i \ominus 1, i, i \oplus 1\}$. Thus, the initial assignment for \mathcal{A}' is given by:

$$H_0(i) = \{a_1^i, \dots, a_n^i\} \cup \{a^{i \ominus 1}, a^i, a^{i \oplus 1}\}$$

In particular, for each $i \in [m]$, $H_0 @ \{i\} = \{a_1^i, \dots, a_n^i\}$ and $H_0 @ \bar{i} = \{a^i\}$. For all other sets X , $H @ X = \emptyset$. We let q_0 and q_F be the initial state and final states of \mathcal{A}' respectively. Moreover, we map each transition $q \xrightarrow{\delta_i} q'$ to $q \xrightarrow{\emptyset, \{i\}} q'$; each $q \xrightarrow{-\delta_i} q'$ to $q \xrightarrow{\{i\}, \emptyset} q'$; and $q \xrightarrow{i} q'$ to a sequence of transitions:

$$q \xrightarrow{\{i\}} \cdot \xrightarrow{\bar{i} \ominus 1 \setminus \{i\}, \bar{i} \ominus 1} \cdot \xrightarrow{\bar{i} \setminus \{i\}, \bar{i}} \cdot \xrightarrow{\bar{i} \oplus 1 \setminus \{i\}, \bar{i} \oplus 1} q'$$

⁶ In [27] we can assume Minsky machines which only branch in the form “if $c=0$ then goto 1; $c--$,” and are therefore deterministic in the above sense, while Schnoebelen’s auxiliary constructions are also deterministic.

The effect of the above is to remove all names appearing precisely in history i , and proceed with a three-name code which distinguishes this reset from any other transition and restores the missing a^j s from history i . Note that the determinacy of \mathcal{A} combined with the way we translate reset transitions, imply strong determinacy of \mathcal{A}' . We can see that \mathcal{A}' simulates the behaviour of \mathcal{A} by storing the value of each counter i as the number of names appearing exactly in its history i . Hence, $(\mathcal{A}, q_0, \vec{v}_0, q_F) \in \text{Reach}$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. \square

We now turn to the questions of universality and language containment. Note first that our machines directly inherit undecidability of these properties from register automata [24].

Lemma 18. *Universality is undecidable for HRAs.*

Nonetheless, as we next show, the above properties are decidable in the deterministic case. In order to simplify our analysis, we shall be reducing HRAs to the following compact form where ϵ -transitions are incorporated inside name-accepting ones. As we show below, no expressiveness is lost by this packed form.

A *packed* $(m, 0)$ -HRA is a tuple $\mathcal{A} = \langle Q, q_0, \delta, H_0, F \rangle$ defined exactly as an $(m, 0)$ -HRA, with the exception that now:

$$\delta \subseteq Q \times \mathcal{P}([m]) \times \mathcal{P}([m]) \times \mathcal{P}([m]) \times Q$$

We shall write $q \xrightarrow{Y;X,X'} q'$ for $(q, Y, X, X', q') \in \delta$. The semantics of such a transition is the same as that of a pair of transitions $q \xrightarrow{Y} \cdot \xrightarrow{X,X'} q'$ of an ordinary HRA. Formally, configurations of packed HRAs are pairs (q, H) , like in HRAs, and the configuration graph of a packed HRA \mathcal{A} like the above is constructed as follows. We set $(q, H) \xrightarrow{a} (q, H')$ if there is some $q \xrightarrow{Y;X,X'} q'$ in δ such that, setting $H_Y = H[Y \mapsto \emptyset]$, we have $a \in H_Y @ X$ and $H' = H_Y[a \text{ in } X']$.

Lemma 19. *Let \mathcal{A} be an $(m, 0)$ -HRA. There is a packed $(m, 0)$ -HRA \mathcal{A}' such that $\mathcal{A} \sim \mathcal{A}'$.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, H_0, F \rangle$. We set $\mathcal{A}' = \langle Q, q_0, \delta', H_0, F' \rangle$ where:

$$\begin{aligned} F' &= \{q' \in Q \mid \exists q \in F, Y. q' \xrightarrow{Y} q \in \delta\} \\ \delta' &= \{(q, Y, X, X', q') \mid q \xrightarrow{Y} \cdot \xrightarrow{X,X'} q' \in \delta\} \end{aligned}$$

Bisimilarity of \mathcal{A} and \mathcal{A}' is witnessed by the identity on configurations, i.e. $R = \{((q, H), (q, H))\}$ is a bisimulation. \square

We shall decide language containment via complementation. In particular, given a deterministic packed HRA \mathcal{A} , the automaton \mathcal{A}' accepting the language $\mathcal{N} \setminus \mathcal{L}(\mathcal{A})$ can be constructed in the analogous way as for deterministic finite-state automata, namely by obfuscating the automaton with all missing transitions and swapping final with non-final states. Finding the missing transitions is easy: for each state q and each set X such that there is no transition of the form $q \xrightarrow{Y;X \setminus Y, X'} q'$ in \mathcal{A} , we add a transition $q \xrightarrow{\emptyset;X,\emptyset} q_F$ to some sink final state q_F .

Lemma 20. *Deterministic packed HRAs are closed under complementation.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, H_0, F \rangle$ be a packed $(m, 0)$ -HRA. Following the above rationale, we construct a packed $(m, 0)$ -HRA $\mathcal{A}' = \langle Q \uplus \{q_F\}, q_0, \delta \cup \delta', H_0, F' \rangle$, where $F' = \{q_F\} \cup (Q \setminus F)$ and δ' is given as follows. For each $q \in Q$ and all X such that there is no $q \xrightarrow{Y; X \setminus Y, X'} q'$ add a transition $q \xrightarrow{\emptyset; X, \emptyset} q_F$ in δ' . In addition, δ' contains a transition $q_F \xrightarrow{[m]; \emptyset, \emptyset} q_F$.

We claim that $\mathcal{L}(\mathcal{A}') = \mathcal{N}^* \setminus \mathcal{L}(\mathcal{A})$. Indeed, if $s \in \mathcal{L}(\mathcal{A}')$ and s is accepted at a state in $Q \setminus F$ then, since \mathcal{A} is deterministic, we have $s \notin \mathcal{L}(\mathcal{A})$. Otherwise, if $s = s'as''$ with a the point where a transition to the sink state is taken then, upon acceptance of s' by \mathcal{A} , a appears precisely in some histories X such that \mathcal{A} has no transition to accept a at that point. Thus, $s \notin \mathcal{L}(\mathcal{A})$.

Conversely, if $s \in \mathcal{N}^* \setminus \mathcal{L}(\mathcal{A})$ then either s induces a configuration in \mathcal{A} which does not end in a final state, or $s = s'as''$ where s' is accepted by \mathcal{A} but at that point a is not a possible transition. We can see that, in each case, $s \in \mathcal{L}(\mathcal{A}')$. \square

Combining the above with the product construction, and using Proposition 16 and the fact that language emptiness can be reduced to language containment, we obtain the following.

Proposition 21. *Language containment and universality are decidable for deterministic HRAs, with non-primitive recursive complexity.*

5 Weakening HRAs

The complexity of HRAs is too high for practical verification purposes. For example, deciding emptiness requires complexity which is not primitive recursive. It is therefore useful to seek for restrictions thereof which allow us to express meaningful properties and, at the same time, remain at feasible complexity. As the complexity of HRAs stems from the fact that they can simulate computations of R-VASSs, our strategy for producing weakenings is to restrict the functionalities of the corresponding R-VASSs. We follow two directions:

- (a) We remove reset transitions. This corresponds to removing counter transfers and resets and drops the complexity of control-state reachability to exponential space.
- (b) We restrict the number of histories to just one. We thus obtain polynomial space complexity as the corresponding counter machines are simply one-counter automata. This kind of restriction is also a natural extension of FRAs with history resets.

Observe that each of the aspects of HRAs targeted above correspond to the distinctive features (a,b) we identified in the Introduction, witnessed by the languages \mathcal{L}_1 and \mathcal{L}_2 respectively. We shall see that each restriction leads to losing the corresponding language.

Our analysis on emptiness for general HRAs from Section 4 is not applicable to these weaker machines as we now need to take registers into account: the simulation of registers by histories is either not possible or not practical for deriving satisfactory bounds. A direct analysis is therefore necessary.

Solving emptiness for each of the weaker versions of HRAs will involve reduction to the name-free algorithmic setting of a counter machine. In both cases, the reduction shall follow the same concept of simulating computations with names *symbolically*. We present the method in full rigour in Appendix A and specialise it for each of the weaker HRAs.

5.1 Non-reset HRAs

We first weaken our automata by disallowing resets. We show that the new machines retain all their closure properties apart from Kleene-star closure. The latter is concretely manifested in the fact that language \mathcal{L}_1 of the Introduction is lost. On the other hand, the emptiness problem reduces in complexity to double exponential space, or exponential space for machines with histories only.

Definition 22. A **non-reset HRA** of type (m, n) is an (m, n) -HRA $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ such that there is no $q \xrightarrow{X} q' \in \delta$.

We call such a machine a non-reset (m, n) -HRA. In an analogous fashion, a **VASS** of m dimensions (an m -VASS) is an R-VASS with no reset transitions. For these machines, control-state reachability is significantly less complex.

Fact 23 ([19,26]) *Control-state reachability for VASSs is EXPSPACE-complete.*

Closure properties Of the closure constructions of Section 3 we can see that union and intersection readily apply to non-reset HRAs. On the other hand, concatenation does use a reset but it can be avoided. More specifically, we add empty transitions from the final states of \mathcal{A}'_1 to the initial state of a version of \mathcal{A}'_2 which keeps the places used by \mathcal{A}'_1 untouched and uses its own separate copy of places, obfuscating its own transitions so as to capture accidental matchings of the legacy names of \mathcal{A}'_1 . Unfortunately, this solution cannot be used for Kleene closure as in each loop the automaton needs to find a fresh copy of its initial configuration, and be able to use it (in the previous construction, the final assignment of \mathcal{A}'_1 is lost). In fact, using an argument similar to that of [5, Proposition 7.2], we can show that the language \mathcal{L}_1 is not recognised by non-reset HRAs and, hence, the latter are not closed under Kleene star. Finally, note that the HRA constructed for the language \mathcal{L}_4 in Example 14 is a non-reset HRA, which implies that non-reset HRAs are not closed under complementation.

Emptiness We next reduce emptiness for non-reset HRAs to control-state reachability for VASSs, using the symbolic construction from Appendix A. The reduction works by mapping each non-empty subset of $[m]$ to a VASS counter. This produces a VASS of exponential size, and of square size if the original non-reset

HRA contained no registers. The latter discrimination is due to the fact that in the general case the status of registers needs to be embedded inside states. For the converse direction, we reduce reachability for a VASS of $2^m - 1$ counters to emptiness for an $(m, 0)$ -automaton: we map each counter to a non-empty subset of $[m]$.⁷

Proposition 24. *Emptiness checking for non-reset HRAs is in 2-EXPSpace. For non-reset HRAs without registers, it is EXPSpace-complete.*

Proof. We reduce from and to VASSs. Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ be a non-reset (m, n) -HRA. We follow the construction from Appendix A to obtain a simulating VASS $\mathcal{A}' = \langle Q', \delta' \rangle$ with $m' = 2^m - 1$ counters. The size N' of each reachability instance $(\mathcal{A}', (q_0, \Sigma(H_0)), \vec{v}_0, (q, \phi))$ simulating emptiness of \mathcal{A} is in $O(2^{\|\mathcal{A}\|^2})$. Now, in the specific case of $n = 0$, the factor $|\Sigma(m, n)|$ trivialises to 1 so, using also the fact that there are no resets:

$$N' \leq |\delta| \cdot 2(2 \log(|Q| \cdot (m' + 1)) + \log(3^{m'})) + \|(q_0, H_0, q)\|$$

Moreover, we do not need to take m' to be $2^m - 1$; it suffices to let m' be the number of non-empty subsets of $[m]$ appearing in δ , that is, all such Y such that there is some $(q, X, X', q) \in \delta$ with $Y \in \{X, X'\}$. It is only these subsets that we need to track with counters. This implies $m' \leq 2|\delta|$ and therefore $N' \in O(\|\mathcal{A}\|^2)$. Thus, from Fact 23 we have that emptiness for \mathcal{A} can in general be checked in double exponential space, and in exponential space if \mathcal{A} has no registers.

Conversely, let $\mathcal{A} = \langle Q, \delta \rangle$ be an m -VASS and $(\mathcal{A}, q_0, \vec{v}_0, q)$ a reachability instance. We construct a non-reset $(m', 0)$ -HRA $\mathcal{A}' = \langle Q', (q_0, \vec{0}), H_0, \delta', \{(q, \vec{0})\} \rangle$, with $m' = \log(m+1)$, as follows. Set $Q' = Q \times \{-1, 0, 1\}^{m-1}$, pick a bijection $\phi : [m] \rightarrow \mathcal{P}_{\neq \emptyset}([m'])$ and an initial assignment H_0 such that, for each i , $|H_0 @ \phi(i)| = v_{0i}$ and set:

$$\begin{aligned} \delta' = & \{((q, \vec{0}), \emptyset, \emptyset, (q', \vec{0})) \mid (q, \vec{0}, q') \in \delta\} \\ & \cup \{((q, \vec{0}), \phi(j), \emptyset, (q', \vec{0}\vec{v})) \mid (q, \vec{0}(-1)\vec{v}, q') \in \delta\} \\ & \cup \{((q, \vec{0}), \emptyset, \phi(j), (q', \vec{v})) \mid (q, \vec{0}(1)\vec{v}, q') \in \delta\} \\ & \cup \{((q, \vec{0}(-1)\vec{v}), \phi(j), \emptyset, (q, \vec{0}\vec{v}))\} \cup \{((q, \vec{0}(1)\vec{v}), \emptyset, \phi(j), (q, \vec{0}\vec{v}))\} \end{aligned}$$

where, in each case, the marked (1) or (-1) appears in the j -th dimension. Thus, we encode each counter i of \mathcal{A} to a set of histories $\phi(i)$ in \mathcal{A}' . Each configuration (q, \vec{v}) of \mathcal{A} is simulated by a configuration $((q, \vec{0}), H)$ of \mathcal{A}' such that $v_i = |H @ \phi(i)|$ for all i . Each transition $q \xrightarrow{\vec{v}} q'$ is then mapped to a sequence of

⁷ Note that such a $(2^m - 1)$ -to- m reduction would not work for general R-VASSs (hence the different reduction in Proposition 17), as resets would misbehave. More specifically, in HRAs a single reset may affect more than one subsets of $[m]$ (e.g. resetting $\{1, 2\}$ clears not only $\{1, 2\}$, but also $\{1\}$ and $\{2\}$) and, in addition, resets cause virtual transfers of names (e.g. resetting history 1 transfers all names appearing precisely in histories 1, 2 to history 2).

transitions $(q, \vec{0}) \xrightarrow{\ell_1} \dots \xrightarrow{\ell_m} (q', \vec{0})$ where, for each i , $\ell_i = (\phi(i), \emptyset)$ if $v_i = -1$; $\ell_i = (\emptyset, \phi(i))$ if $v_i = 1$; and ℓ_i is empty otherwise (actually, we drop it altogether). In particular, to specify the transitions that need to be taken between $(q, \vec{0})$ and $(q', \vec{0})$ we use the extra vector component of states. E.g. a transition $q \xrightarrow{100(-1)1} q'$ is mapped to:

$$(q, 0000) \xrightarrow{\emptyset, \phi(1)} (q', 00(-1)1) \xrightarrow{\phi(4), \emptyset} (q', 0001) \xrightarrow{\emptyset, \phi(5)} (q', 0000)$$

By construction, we have $(\mathcal{A}, q_0, \vec{v}_0, q) \in \text{Reach}$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Moreover,

$$\|\mathcal{A}'\| = |\delta'| \cdot (2 \log |Q'| + 2 \log(m+1)) + \|(q_0, \vec{0}, H_0, q, \vec{0})\|$$

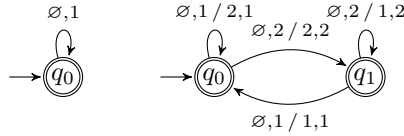
and $|\delta'| \leq |\delta| \cdot m$, from which we obtain $\|\mathcal{A}'\| \in O(\|\mathcal{A}\|^2)$. \square

Non-reset HRAs without registers We now show that non-reset HRAs with histories only are equi-expressive as general non-reset HRAs. The equivalence we prove is weaker than the one we proved for general HRAs: we show language equivalence rather than bisimilarity. Our proof below is based on the *colouring technique* of [5]. Before we proceed with the proof, let us first demonstrate the technique through an example.

Example 25. It is easy to see that the following language

$$\mathcal{L}_5 = \{a_1 \dots a_n \in \mathcal{N}^* \mid \forall i. a_i \neq a_{i+1}\}$$

is recognised by the $(0, 1)$ -HRA on the left below. What is perhaps not as clear is that the $(2, 0)$ -HRA on the right, call it \mathcal{A} , accepts the same language.



Note first that, by construction, it is not possible for \mathcal{A} to accept the same name in two successive transitions: if we write (X, X') for the labels of incoming transitions to q_0 and (Y, Y') for the outgoing, we cannot match any X' with some Y , and similarly for q_1 . This shows $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_5$. To prove the other inclusion, we need to show that for every word $w = a_1 \dots a_n \in \mathcal{L}_5$ there is an accepting run in \mathcal{A} . For this, it suffices to find a sequence ℓ_1, \dots, ℓ_n of labels from the set $\{(\emptyset, 1), (\emptyset, 2), (1, 1), (1, 2), (2, 1), (2, 2)\}$, say $(\ell_i = (X_i, X'_i))$, satisfying:

1. For any i , $X'_i \neq X_{i+1}$.
2. If $a_i = a_j$, $i < j$, and for no $i < k < j$ do we have $a_i = a_k$ then $X'_i = X_j$.
3. For any i , if $a_i \neq a_j$ for all $j < i$ then $X_i = \emptyset$.

The first condition ensures that the sequence corresponds to a valid transition sequence in \mathcal{A} , and the other two that the sequence accepts the word $w =$

$a_1 \cdots a_n$. Conditions 1 and 2 determine dependencies between the choices of left and right components in ℓ_i s. Let us attach to w dependency pointers as follows: attach a pointer of type 1 (dependency right-to-left) from each a_i to its next occurrence in w , say a_j ; from each a_{i+1} attach a type 2 pointer (dependency left-to-right) to a_i . Now note that, as there is no cycle in w which alternates between type 1 and type 2 pointers, it is always possible to produce a valid sequence ℓ_1, \dots, ℓ_n .

We now show the general result. We assume automata with their registers initially empty – the general case can be captured by first applying a construction like $\mathcal{A}\text{fix } w$ of Lemma 11 (the lemma introduces new registers for storing the initial names, but we can as well use new histories for the same purpose). The proof is presented in Appendix B.

Proposition 26. *For each (n, m) -non-reset HRA \mathcal{A} with initially empty registers there is an $(n+3m, 0)$ -non-reset HRA \mathcal{A}' such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

5.2 Unary HRAs

Our second restriction concerns allowing resets but bounding the number of histories to just 1. Thus, these automata are closer to the spirit of FRAs and, in fact, extend them by rounding up their history capabilities. We show that these automata require polynomial space complexity for emptiness and retain all their closure properties apart from intersection. The latter is witnessed by failing to recognise \mathcal{L}_2 from the Introduction. We can see that extending this example to multiple interleavings we can show that intersection is in general incompatible with bounding the number of histories.

Definition 27. *A $(1, n)$ -HRA is called a **unary HRA** of n registers.*

In other words, unary HRAs are extensions of FRAs where names can be selectively inserted or removed from histories and, additionally, histories can be reset. These capabilities give us in fact a strict extension.

Example 28. The automata used in Example 3 for \mathcal{L}_1 and \mathcal{L}_3 were unary HRAs. Note that neither of these languages is FRA-recognisable. On the other hand, the language \mathcal{L}_2 is not recognisable by unary HRAs.

For suppose $\mathcal{L}_2 = \mathcal{L}(\mathcal{A})$ for some unary HRA \mathcal{A} of n registers and let

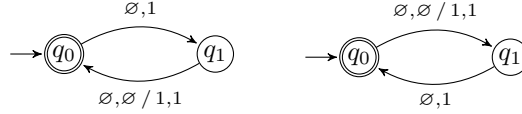
$$w = a_1 b_1 \cdots a_k b_k b_1 a_1 \cdots b_k a_k$$

for $k = n+1$ and some pairwise distinct names $a_1, b_1, \dots, a_k, b_k$. As $w \in \mathcal{L}_2$, there is a path, say p , in \mathcal{A} which accepts w . We divide p as $p_1 p_2$ with p_2 accepting the second half of w . Let $\hat{p} = \hat{p}_1 \hat{p}_2$ be the corresponding configuration path and let (q', H') be the first configuration in \hat{p}_2 . We set $S = \{a_1, b_1, \dots, a_k, b_k\} \setminus \{a \mid a \in H'(i) \wedge i > 1\}$ and do a case analysis on the labels of the form (X, X') which appear in p_2 and accept names from S . Since names in S do not appear in any $H'(i)$, for $i > 0$, it must be that each such X is either $\{1\}$ or \emptyset . We have the following cases.

- There are two such labels, say $(\{1\}, X_i)$ and $(\{1\}, X_j)$, accepting names a_i and b_j respectively. But this would imply that \mathcal{A} also accepts w' , where w' is w with these occurrences of a_i and b_j swapped, contradicting $\mathcal{L}(\mathcal{A}) = \mathcal{L}_2$ (as $w' \notin \mathcal{L}_2$).
- There are two such labels, say (\emptyset, X_i) and (\emptyset, X_j) , accepting names a_i and b_j respectively. In order for \mathcal{A} not to accept w' (w' as above), it is necessary that a reset transition with label $Y \ni 1$ occurs between the two transitions. Suppose $i < j$. Then, since $k > n$, there is a name $a_{i'}$ which does not appear in any place after clearing Y . Thus, (\emptyset, X_j) can accept $a_{i'}$ and complete the path p by accepting a word $w' \notin \mathcal{L}_2$. Dually if $j \leq i$.
- Each $a_i \in S$ is accepted by a label $(\{1\}, X')$, and each $b_j \in S$ by a label (\emptyset, X') . Let $a_i \in S$ be the last such accepted in p_2 . This means that the rest of the path has length at most $2n$. Therefore, since $k > n$, there is a $b_j \in S$ accepted in p_2 before a_i . Let (q, H) be the configuration just before accepting b_j . In order for \mathcal{A} not to accept any $a_{i'}$ at that point, it must be that all $a_{i'} \in S$ appear in H . Since $|S| > n + 1$, there exists $a_{i'} \in H(1) \cap S$ such that $a_{i'} \neq a_i$. But then, the transition accepting a_i can accept $a_{i'}$ instead and lead to acceptance of a word $w' \notin \mathcal{L}_2$.

We therefore reach a contradiction in every case.

Closure properties The closure constructions of Section 3 readily apply to unary HRAs, with one exception: intersection. For the latter, we simply observe that $\mathcal{L}_2 = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, where \mathcal{A}_1 and \mathcal{A}_2 are the following unary $(1, 0)$ -HRAs, with empty initial assignments.



We can see that their corresponding languages are:

$$\begin{aligned}\mathcal{L}(\mathcal{A}_1) &= \{a_1 a'_1 \cdots a_n a'_n \in \mathcal{N}^* \mid a_1 \cdots a_n \in \mathcal{L}_0\} \\ \mathcal{L}(\mathcal{A}_2) &= \{a_1 a'_1 \cdots a_n a'_n \in \mathcal{N}^* \mid a'_1 \cdots a'_n \in \mathcal{L}_0\}\end{aligned}$$

On the other hand, unary HRAs are not closed under complementation as one can construct unary HRAs accepting $\overline{\mathcal{L}(\mathcal{A}_1)}$ and $\overline{\mathcal{L}(\mathcal{A}_2)}$, and then take their union to obtain a unary HRA for $\overline{\mathcal{L}_2}$.

Emptiness In the case of just one history, the results on TR-VASS reachability [27,10] we used in Section 4 provide rather rough bounds. It is therefore useful to do a direct analysis. We examine control-state reachability for R-VASSs of 1 dimension. Note that these machines can be seen as close relatives to several other formalisms, like one-counter automata or pushdown automata on a one-letter alphabet. To the best of our knowledge, though, there has been no direct attack of state reachability for R-VASSs of 1 dimension. Our analysis below (proof in Appendix B) yields square minimal-path length.

Lemma 29. *Control-state reachability for R-VASSs of dimension 1 can be decided in $\text{SPACE}(\log^2 N)$.*

We can now proceed with our main result for unary HRAs. Each such automaton \mathcal{A} will be reduced to an R-VASS \mathcal{A}' with one counter, following the method presented in Appendix A.

Proposition 30. *Emptiness of unary HRAs can be decided in $\text{SPACE}((N \log N)^2)$.*

Proof. Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ be a unary HRA of n registers and let $N = \|\mathcal{A}\|$. Following the construction from Appendix A we obtain a simulating R-VASS $\mathcal{A}' = \langle Q', \delta' \rangle$ with 1 counter such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there is (q, ϕ) with $q \in F$ such that $(\mathcal{A}', (q_0, \Sigma(H_0)), |H_0(1)|, (q, \phi)) \in \text{Reach}$. By Lemma 29, the latter can be decided in squared logarithmic space. Note that this does not require actually constructing \mathcal{A}' : we can apply the algorithm of Lemma 29 on the fly, using only the space required for running it. Moreover, we can run each instance (for different (q, ϕ) 's) in the same space. Now, since there is only one history, the induced size of \mathcal{A}' is relatively smaller. In particular, setting $M = |\Sigma(1, n)|$, we have $\log M \leq n + n \log n + 1 \leq N \cdot \log N$ so the size of $(\mathcal{A}', (q_0, \Sigma(H_0)), |H_0(1)|, (q, \phi))$ is:

$$\begin{aligned} N' &= |\delta'| \cdot (2 \log |Q'| + 2) + \|(q_0, H_0, q, \phi)\| \\ &\leq |\delta| \cdot 2M \cdot (2 \log(|Q| \cdot M \cdot 2) + 2) + \|(q_0, H_0, q)\| + \log M \end{aligned}$$

and hence $N' \leq 2^{N \cdot \log N} \cdot N^2 \cdot \log N + N \cdot \log N$. Thus, by Lemma 29, we can decide emptiness of \mathcal{A} in $\text{SPACE}((N \log N)^2)$. \square

6 Connections with existing formalisms

We have already seen that HRAs strictly extend FRAs. In this section we shall draw connections between HRAs and an automata model over infinite alphabets in the limits of decidability, called *Data Automata (DA)*, introduced in [6] in the context of XML database theory and model checking. DAs operate on *data words*, that is, over finite sequences of elements from $\mathcal{S} \times \mathcal{N}$, where \mathcal{S} is a finite set of *data tags* and \mathcal{N} incarnates an infinite set of *data values* (but we shall call them *names*). A DA operates in two stages which involve a transducer automaton and a finite-state automaton respectively. Both automata operate on the tag projection of the input word, with the second-stage automaton focussing on data tags accompanied by the same data value.

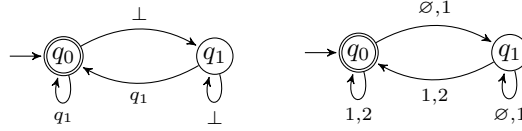
For the rest of our discussion we shall abuse data words and treat them simply as strings of names, neglecting data tags. This is innocuous since there are straightforward translations between the two settings.⁸ An equivalent formalism to DAs which is closer to our framework is the following [5].

⁸ A string of names is the same as a data word over a singleton set of data tags; while data tags can be simulated by names in registers of the initial configuration which do not get moved nor copied during the computation.

Definition 31. A **Class Memory Automaton (CMA)** is a tuple $\mathcal{A} = \langle Q, q_0, \phi_0, \delta, F_1, F_2 \rangle$ where Q is a finite set of states, $q_0 \in Q$ is initial, $F_1 \subseteq F_2 \subseteq Q$ are sets of final states and the transition relation is of type $\delta : Q \times (Q \cup \{\perp\}) \rightarrow \mathcal{P}(Q)$. Moreover, ϕ_0 is an initial class memory function, that is, a function $\phi : \mathcal{N} \rightarrow Q \cup \{\perp\}$ with finite domain $\{a \mid \phi(a) \neq \perp\}$ is finite).

The semantics of a CMA \mathcal{A} like the above is given as follows. Configurations of \mathcal{A} are pairs of the form (q, ϕ) , where $q \in Q$ and ϕ a class memory function. The configuration graph of \mathcal{A} is constructed by setting $(q, \phi) \xrightarrow{a} (q', \phi')$ just if there is $(q, \phi(a), q') \in \delta$ and $\phi' = \phi[a \mapsto q']$. The initial configuration is (q_0, ϕ_0) , while a configuration (q, ϕ) is accepting just if $q \in F_1$ and, for all $a \in \mathcal{N}$, $\phi(a) \in F_2 \cup \{\perp\}$.

Thus, CMAs resemble HRAs in that they store input names in “histories”, only that histories are identified with states: for each state q there is a corresponding history q (note notation overloading), and a transition which accepts a name a and leads to a state q must store a in the history q . Moreover, each name appears in at most one history (hence the type of ϕ) and, moreover, the finality conditions for configurations allow us to impose that all names appear in specific histories, if they appear in any. For example, here is a CMA (left below, with $F_1 = F_2 = \{q_0\}$) which recognises the language $\bar{\mathcal{L}}_4$.



Each name is put in history q_1 when seen for the first time, and to q_0 when seen for the second time. The automaton accepts if all its names are in q_0 . This latter condition is what makes the essential difference to HRAs, namely the capability to check where the names reside for acceptance. For example, the HRA on the right above would accept the same language were we able to impose the condition that accepting configurations (q, H) satisfy $a \in H@2$ for all names $a \in \bigcup_i H(i)$.

The above example proves that HRAs cannot express the same languages as CMAs. Conversely, as shown in [5, Proposition 7.2], the fact that CMAs lack resets does not allow them to express languages like, for example, $\mathcal{L}_1 = (\mathcal{L}_{a_0})^*$ where:

$$\mathcal{L}_{a_0} = \{aw \in \mathcal{L}_0 \mid w \in \mathcal{N}^* \wedge a = a_0\}$$

In the latter sections of [5] several extensions of CMAs are considered, one of which does involve resets. However, the resets considered there do not seem directly comparable to the reset capability of HRAs.

On the other hand, a direct comparison can be made with non-reset HRAs. We already saw in Proposition 26 that, in the latter formalism, histories can be used for simulating register behaviour. In the absence of registers, CMAs differ from non-reset HRAs solely in their constraint of relating histories to states (and their termination behaviour, which is more expressive). As the latter can be easily counterbalanced by obfuscating the set of states, we obtain the following.

Proposition 32. *For each non-reset HRA \mathcal{A} there is a CMA \mathcal{A}' such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

7 Further directions

We see several further directions of this work. A first direction is examining the decidability of bisimilarity. Although it is known that bisimilarity is undecidable for Petri nets [14], the version which seems of relevance towards an undecidability argument for HRA-bisimilarity is that of *visibly* counter automata with labels, i.e. automata which accept labels at each transition, and the action of each transition is determined by the accepted label. The latter problem is not known to be decidable. On the other hand, it seems reasonable to consider further extensions of HRAs with additional expressiveness added by constrained tests for zero, for example as in [11]. Finally, an avenue we would like to pursue is the application of automata with histories in runtime verification, in the spirit of [13]. Although the complexity results derived in this paper may seem discouraging at first, they are based on quite specific representations of hard problems. In practice, we expect programs to yield automata of simpler complexities. Experience with tools based on coverability of TR-VASSs, like e.g. BFC [16], positively testify in that respect.

A Reasoning about emptiness symbolically

In this section we describe a method for reducing emptiness for HRAs to control-state reachability for TR-VASSs. For convenience, we shall consider a specific subclass of HRAs which encompasses the weak HRAs we examine in Section 5, and for which the corresponding TR-VASSs are just R-VASSs. More specifically, we examine HRAs $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ such that:

$$q \xrightarrow{X} q' \in \delta \wedge X \cap [m] \neq \emptyset \implies [m] \subseteq X$$

We can see that in the above machines it is not possible to virtually transfer names between histories. Therefore, all possible transfers are additions of single names, or literal resets.

Before we proceed with our analysis, let us fix the way we compute the size of our machines. For a set X , we write $|X|$ for its cardinality. For a structure X , we write $\|X\|$ for its size. We assume that in general we need $\log(k)$ space to encode any i from a set $[k]$.

HRA Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ be an (m, n) -HRA. We represent \mathcal{A} by encoding δ, q_0, H_0 and F , where δ is encoded as an array of tuples of the form (q, ℓ, q') with $\ell \in \mathcal{P}([m+n]^2 \cup \mathcal{P}([m+n]))$. We have $\|\delta\| = |\delta| \cdot (2 \log |Q| + \log(2^{2(m+n)} + 2^{m+n}))$ and so:

$$\begin{aligned} \|\mathcal{A}\| &= \|\delta\| + \|(q_0, H_0)\| + \|F\| \\ &= |\delta| \cdot (2 \log |Q| + \log(2^{2(m+n)} + 2^{m+n})) + \|(q_0, H_0)\| + \|F\| \end{aligned}$$

where $\|F\| = |F| \cdot \log |Q|$. A rough lower bound of the above is:

$$\|\mathcal{A}\| \geq 2|\delta| \cdot \log |Q| + 2|\delta| \cdot (m+n) + \|(q_0, H_0)\| + \|F\|$$

R-VASS Let $\mathcal{A} = \langle Q, \delta \rangle$ be an R-VASS of m dimensions. Encoding δ as an array with entries of the form (q, ℓ, q') , where $\ell \in \{-1, 0, 1\}^m \cup [m]$, we have that the size of \mathcal{A} is:

$$\|\mathcal{A}\| = \|\delta\| = |\delta| \cdot (2 \log |Q| + \log(3^m + m))$$

The size $\|(\mathcal{A}, q_0, i_0, q_F)\|$ of an input to state-reachability is:

$$|\delta| \cdot (2 \log |Q| + \log(3^m + m)) + \|(q_0, i_0)\| + \|q_F\|$$

Skeletons Checking language emptiness for a given HRA boils down to determining the existence of a *symbolic accepting path*, i.e. a sequence of δ -steps from the initial state to some final state, which is also *realisable* by the HRA, i.e. it leads to a corresponding configuration path. Finding symbolic paths from initial to final states is easy: one just needs to decide emptiness for the finite-state automaton underlying the HRA in examination. On the other hand, resolving whether a path is realisable is more demanding: in order for a transition $q \xrightarrow{X, X'} q'$ to be realisable, it is necessary that the automaton be at a configuration H such that there exists a name appearing precisely in places X ; put otherwise, $|H @ X| > 0$.

We address the above by constructing counter machines which operate symbolically and carry along during computation (inside their states and counters) the minimal amount of information necessary for resolving whether a transition of the modelled HRA can be taken. This information corresponds to a symbolic encoding of actual assignments. For each assignment H , it consists of:

- (i) A partition of the set of registers according to whether they contain the same name in H , and a function assigning to each history those registers whose contents appear in that history.
- (ii) For each set of histories X , a counter which stores the number of names which appear exactly in X .

We call part (i) the *skeleton* of the assignment, and part (ii) its *counters*. Consider, for example, the (1, 4)-HRA assignments:

$$\begin{aligned} H_1 &= \{(1, \{b\}), (2, \{a\}), (3, \emptyset), (4, \{b\}), (5, \{a\})\} \\ H_2 &= \{(1, \{a, b, c\}), (2, \{d\}), (3, \emptyset), (4, \{a\}), (5, \{d\})\} \end{aligned} \tag{1}$$

Their skeletons coincide as, in both assignments:

- the first and fourth registers (places 2 and 5) contain the same name, the third register (place 4) contains a different one, and the second register (place 3) is empty;
- the history (place 1) contains the name of the third register, and no other names from the registers.

On the other hand, their counters differ: the former leads to a counter value 0 (no names appearing only in the history), while the latter leads to 2.

Formally, for each (m, n) we set $\Sigma(m, n)$ to be the set of all functions $\phi : [m+n] \rightarrow \mathcal{P}([n])$, such that:

- For all $i > m$, $|\phi(i)| \leq 1$ and, moreover, the values for $\phi(i)$ are picked in the following sequential manner: if $\phi(i) = \{j\}$ then either $\phi(i') = \{j\}$ for some $m < i' < i$, or j is the least element of $[n]$ which has not appeared in $\phi(m+1), \dots, \phi(i-1)$.
- For all $i \leq m$, $\phi(i) \subseteq \bigcup_{i' > m} \phi(i')$.

We call elements of $\Sigma(m, n)$ *skeletons*. For each skeleton ϕ , its restriction to indices greater than m represents a partition of the n registers: registers i and j are in the same part if $\phi(m+i) = \phi(m+j) \neq \emptyset$; register i is empty if $\phi(m+i) = \emptyset$. On the other hand, for each $i \leq m$, $\phi(i)$ contains the index $\phi(m+j)$ just if the name of the j -th register appears in the i -th history. For example, both configurations in (1) have the skeleton $(\{2\}, \{1\}, \emptyset, \{2\}, \{1\})$.

Note that, because of the canonical selection of indices for skeleton values, each assignment H has a unique skeleton, which we denote by $\Sigma(H)$. Moreover,

$$|\Sigma(m, n)| \leq 2^{mn} \cdot (n+1)! \leq 2^{mn+n \log n+1} \quad (2)$$

which is slightly better than the $2^{n(m+n)}$ we would have obtained had we taken skeletons to be all functions in $[m+n] \rightarrow \mathcal{P}([n])$. Let us now describe how do we update skeletons so that they mimic assignment updates. For each ϕ and $X \subseteq [m+n]$, we set $\phi @ X = \bigcap_{i \in X} \phi(i) \setminus \bigcup_{i \notin X} \phi(i)$. In the specific case of $X = \emptyset$ we slightly abuse this and set $\phi @ \emptyset = \{0\}$. Thus, the $@$ operator mimics the $@$ operator for assignments: if $\phi @ X = \{j\}$ and $\phi = \Sigma(H)$, for some assignment H , then there is a name a such that $H @ X = \{a\}$. In such a case, we write $\phi[j \text{ in } X']$ for $\Sigma(H[a \text{ in } X'])$, for any X' . Finally, for each ϕ representing some H and $X \subseteq [m+n]$ we write $\phi[X \mapsto \emptyset]$ for $\Sigma(H[X \mapsto \emptyset])$. This completes the definition of skeleton updates.

Simulating R-VASSs We now proceed to the concrete construction of counter machines, in the form of R-VASSs, which simulate HRAs for emptiness. Suppose $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ is an (m, n) -HRA and let $X_1, \dots, X_{m'}$ be an enumeration of the non-empty subsets of $[m]$ (which require each a counter for the symbolic representation of assignments), so $m' = 2^m - 1$. We construct a simulating R-VASS $\mathcal{A}' = \langle Q', \delta' \rangle$ with m' dimensions in which states are equipped with skeletons. At skeleton ϕ , each $q \xrightarrow{X, X'} q' \in \delta$ shall be mapped to a sequence of transitions in \mathcal{A}' , according to the following rules.

- If $X \neq \emptyset$ and $\phi @ X = \emptyset$ then the sequence is void.
- Otherwise, if $X = X_j$ then the first transition is a decrement of the j -th counter.
- If $X' = X_i$ then the next transition is an increment of the i -th counter.

- If $X' \not\subseteq [m]$ then the next transition increments each counter i such that there is a set Y_i with $\phi @ Y_i \neq \emptyset$ and $X_i = Y_i \setminus (X' \setminus [m])$. This reconciles for the fact that the registers in X' will be overwritten and therefore their names transferred.

On the other hand, at the same ϕ , a transition $q \xrightarrow{X} q' \in \delta$ shall be mapped to a transition in \mathcal{A}' , according to the rules:

- If $[m] \subseteq X$ then the sequence is just a reset of all counters.
- Otherwise, $[m] \cap X = \emptyset$ and, therefore, transitions increasing all counters according to the last point above will occur.

In each case, the skeleton is updated according to X, X' .

Formally, we set $Q' = Q \times \Sigma(m, n) \times (\{0\} \cup [m'])$, where the last component is an index used to facilitate breaking each transition of δ into potentially two transitions. We include in δ' the following transitions. For each $(q, \phi, 0) \in Q'$ and $q \xrightarrow{X, X'} q' \in \delta$:

- if $\phi @ X = \{j\}$ and $X' = X_i$, add $(q, \phi, 0) \xrightarrow{\delta_i} (q', \phi', 0)$ where $\phi' = \phi[j \text{ in } X']$;
- if $\phi @ X = \{j\}$ and $X' \notin \{X_1, \dots, X_{m'}\}$, add $(q, \phi, 0) \xrightarrow{\vec{v}} (q', \phi', 0)$ where $\phi' = \phi[j \text{ in } X']$ and $v_j = 1$ for all j such that there is Y_j with $\phi @ Y_j \neq \emptyset$ and $X_j = Y_j \setminus (X' \setminus [m])$, and $v_j = 0$ otherwise;
- if $X = X_i$ and $X' = X_j$, add a sequence $(q, \phi, 0) \xrightarrow{-\delta_i} (q', \phi', j) \xrightarrow{\delta_j} (q', \phi', 0)$ where $\phi' = \phi[0 \text{ in } X']$;
- if $X = X_i$ and $X' \notin \{X_1, \dots, X_{m'}\}$, add a sequence $(q, \phi, 0) \xrightarrow{\vec{v}[i \mapsto -1]} (q', \phi', v_i) \xrightarrow{\vec{v}'} (q', \phi', 0)$ where $\phi' = \phi[0 \text{ in } X']$ and $v_j = 1$ for all j such that there is Y_j with $\phi @ Y_j \neq \emptyset$ and $X_j = Y_j \setminus (X' \setminus [m])$, and $v_j = 0$ otherwise, and $\vec{v}' = \vec{0}[i \mapsto v_i]$.

Moreover, for each $q \xrightarrow{X} q' \in \delta$:

- if $[m] \cap X = \emptyset$ then add $(q, \phi, 0) \xrightarrow{\vec{v}} (q, \phi[X \mapsto \emptyset], 0)$ where $v_j = 1$ for all j such that there is Y_j with $\phi @ Y_j \neq \emptyset$ and $X_j = Y_j \setminus (X' \setminus [m])$, and $v_j = 0$ otherwise;
- if $[m] \subseteq X$ then add $(q, \phi, 0) \xrightarrow{j} (q, \phi[X \mapsto \emptyset], 0)$ where $X_j = [m]$.

Finally, we set $\vec{v}_0 = \{(j, |H_0 @ X_j|) \mid j \in [m']\}$.

Thus, \mathcal{A}' symbolically simulates the transition behaviour of \mathcal{A} and updates at each transition its skeleton and counters in such a way that, for each configuration (q, H) of \mathcal{A} , \mathcal{A}' is in the configuration $(q, \phi, 0, \vec{v})$ with $\Sigma(H) = \phi$ and $|H @ X_j| = v_j$ for each $j \in [m']$. We can therefore show the following.

Lemma 33. *For $\mathcal{A}, \mathcal{A}'$ as above, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there is $(q, \phi) \in Q'$ with $q \in F$ such that $(\mathcal{A}', (q_0, \Sigma(H_0)), \vec{v}_0, (q, \phi)) \in \text{Reach}$.*

Let us now compare the size of each of the constructed reachability instances to that of the HRA we started from. Note first that in encoding H_0 we need only encode its internal structure; the specific names appearing in it are of no importance (e.g. we can assume we have a canonical way to produce them). As the structure of H_0 is precisely $(\Sigma(H_0), \vec{v}_0)$, we have $\|H_0\| = \|(\Sigma(H_0), \vec{v}_0)\|$. Thus, the size N' of the instance $(\mathcal{A}', (q_0, \Sigma(H_0)), \vec{v}_0, (q, \phi))$ is:

$$\begin{aligned} N' &= |\delta'| \cdot (2 \log |Q'| + \log(3^{m'} + m')) + \|(q_0, H_0, q, \phi)\| \\ &\leq |\delta| \cdot 2|\Sigma(m, n)| \cdot (2 \log(|Q| \cdot |\Sigma(m, n)| \cdot (m' + 1)) + \log(3^{m'} + m')) \\ &\quad + \|(q_0, H_0, q)\| + \log |\Sigma(m, n)| \end{aligned}$$

where we use the fact that to any transition in δ correspond at most $|\Sigma(m, n)|$ transitions in δ' (one for each skeleton). Now, $\log |\Sigma(m, n)| \leq mn + n \log n + 1 \leq N^2$, where $N = \|\mathcal{A}\|$. Hence, after rough simplifications, from the above we obtain $N' \leq 2^{N^2+N} \cdot N + N^2$.

B Deferred proofs

Proof (Proof of Lemma 11). We construct $\mathcal{A} \text{ fix } w = \langle Q', q'_0, H'_0, \delta', F' \rangle$ as follows. First, we insert/move all names of w to the new registers (places $[m+n+1, m+n+k]$), i.e. we set $H'_0(i) = H_0(i) \setminus \{a_1 \cdots a_k\}$ for all $i \in [m+n]$, and $H'_0(m+n+i) = \{a_i\}$ for each $i \in [k]$. The role of the new registers is to constantly store the names in w and act on the behalf of other places when the latter intend to use those names: during computation, whenever an a_i is captured by a transition of the initial automaton \mathcal{A} , in $\mathcal{A} \text{ fix } w$ it will be instead simulated by a transition involving the new registers. In order for the simulation to be accurate, we shall inject inside states information specifying the *intended* location of the a_i s in the places of \mathcal{A} . Thus, the states of the new automaton are pairs (q, f) , where $q \in Q$ and f is a function recording, for each of the new registers, where would the name of the register appear in the original automaton \mathcal{A} . That is,

$$Q' = Q \times \{f : [k] \rightarrow \mathcal{P}([m+n]) \mid \forall j \neq j'. f(j) \cap f(j') \subseteq [m]\}$$

while $q'_0 = (q_0, \{(i, \{j \mid a_i \in H_0(j)\}) \mid i \in [k]\})$ and $F' = \{(q, f) \in Q' \mid q \in F\}$. Finally, δ' operates just like δ albeit taking into account the f 's of states to figure out the intended positions of the a_i s and, at the same time, update the f 's after each transition. We therefore include in δ' precisely the following transitions.

Below we write i° for $m+n+i$. For each $(q, f) \in Q'$ and $q \xrightarrow{X, X'} q' \in \delta$,

- add a transition $(q, f) \xrightarrow{X, X'} (q', f)$;
- if $f(i) = X$ for some i then add $(q, f) \xrightarrow{\{i^\circ\}, \{i^\circ\}} (q', f')$ where $f' = f[i \mapsto X']$;

Moreover, for each $q \xrightarrow{X} q' \in \delta$ include $(q, f) \xrightarrow{X} (q', f')$ where $f' = \{(j, f(j) \setminus X) \mid i \in [k]\}$.

Following the above line of reasoning, we can show that the relation

$$\{((q, H), (q, f, H')) \mid \forall i \in [m+n]. H(i) = H'(i) \cup \{a_j \mid i \in f(j)\}\}$$

with $(q, H), (q, f, H')$ reachable configurations, is a bisimulation. \square

Proof (Lemma 29). Let $\mathcal{A} = \langle Q, \delta \rangle$ be an R-VASS of 1 dimension. We first state two rather straightforward facts about \mathcal{A} .

Fact 1. \mathcal{A} is up-monotonic: if $(q, i) \twoheadrightarrow (q', i')$ is a configuration path of \mathcal{A} then, for each $k > 0$, there is a path $(q, i+k) \twoheadrightarrow (q', i'')$ of the same length.

Fact 2. \mathcal{A} is down-monotonic: if $(q, i) \twoheadrightarrow (q', i')$ is a configuration path of \mathcal{A} in which there are no reset edges and the counter never becomes less than k , some $k > 0$, then there is a path $(q, i-k) \twoheadrightarrow (q', i'')$ of the same length.

Now, let $(\mathcal{A}, q_0, i_0, q_F)$ be an instance for **Reach** and let p be a configuration path from (q_0, i_0) to (q_F, i_F) , some i_F , such that p is of least length among all paths leading to some (q_F, i) . By Fact 1, we have that p is non-decreasing: if (q, i') appears after (q, i) in p and $i' \leq i$ then we can circumvent the entire subpath between (q, i) and (q, i') . Now suppose $(q, i+k)$ appears after (q, i) in p , some $k > 0$. By Fact 2 and minimality of p , there must be a configuration $(q', k-1)$ after $(q, i+k)$ in p such that there are no reset edges between $(q, i+k)$ and $(q', k-1)$. Let p' be a subpath of p of the form $(q, i+k), (q'', i+k-1), \dots, (q', k-1)$.⁹ Since p' is non-decreasing, all its states are different and hence, as p' has length $i+2$, we have $i+2 \leq |Q|$, i.e. $i \leq |Q|-2$. This gives us a bound on the counter value of any state that can be repeated in p . Thus, each state can appear in p at most $|Q|$ times, with counter values $0, 1, 2, 3, \dots, |Q|-2, j$ (last occurrence can have any value). This implies that the length of p is at most $|Q|^2$ and that in p the counter does not exceed the value $i_0 + |Q|^2 - 1$.

We can therefore check $(\mathcal{A}, q_0, i_0, q_F) \in \text{Reach}$ as follows. Note first that, by Facts 1,2 and the fact that the length of minimal reaching path is at most $|Q|^2$, if $i_0 \geq |Q|^2 - 1$ then it suffices to check $(\mathcal{A}, q_0, |Q|^2-1, q_F) \in \text{Reach}$. Thus, we need only check $(\mathcal{A}, q_0, N_0, q_F) \in \text{Reach}$ with $N_0 = \min(i_0, |Q|^2-1)$. We do this by non-deterministically computing $|Q|^2$ consecutive configurations and checking whether any of them is final. We only store the current configuration and a counter bounded by $|Q|^2$. Thus, we require space $\log |Q| + \log(N_0 + |Q|^2 - 1)$ for the configuration and $\log |Q|^2$ for the counter so, in total, less than $\log |Q| + \log(2|Q|^2) + \log |Q|^2$. Since $N = \|(\mathcal{A}, q_0, N_0, q_F)\| \geq |Q| \cdot \log |Q|$, we require space $O(\log N)$. By Savitch's theorem, we get $\text{SPACE}(\log^2 N)$. \square

Proof (Proposition 26). For simplicity, and in order to make our argument cleaner, the proof we present here assumes that names are kept in at most one register at a time, and in particular transitions have singletons for labels as far as registers are concerned. The same technique, though, applies to the general case modulo some extra technical nuances. For the same reasons, we shall assume $n = 0$. We shall simulate each register of \mathcal{A} by 3 histories in \mathcal{A}' : each register i in \mathcal{A} will have counterparts i_r, i_b, i_y in \mathcal{A}' .¹⁰ Whenever a name a is assigned to register i (in \mathcal{A}), in \mathcal{A}' it is assigned to one of i_r, i_b, i_y . More precisely, it is assigned to i_r if the next time the name appears in the computation of \mathcal{A} is by

⁹ Here by subpath we mean a sequence of nodes from p in the same order as in p .

¹⁰ The indices b and y stand for *black* and *yellow*, in accordance to [5], and r stands for *read*.

reading register i . Otherwise, the next occurrence of a can only happen after the contents of register i are rewritten, and a is assigned again to some register j as a locally fresh name; in the latter case, the name will be stored in either of i_b, i_y . The reason we need two histories for the latter case is so that we can ensure that \mathcal{A}' does not select a locally non-fresh name when it is expected to accept instead a locally fresh one. This can already be seen in Example 30, and will become clearer below.

Let $\mathcal{A} = \langle Q, q_0, H_0, \delta, F \rangle$ with $H_0(i) = \emptyset$ for all $i \in [m]$. We set $\mathcal{A}' = \langle Q', q'_0, H'_0, \delta', F' \rangle$, a non-reset HRA of type $(3m, 0)$, where

$$Q' = Q \times ([m] \rightarrow \{\emptyset, r, b, y\}),$$

$q'_0 = (q_0, \{(j, \emptyset) \mid j \in [3m]\})$, $H'_0 = \{(j, \emptyset) \mid j \in [3m]\}$ and $F' = \{(q, f) \in Q' \mid q \in F\}$. The extra component f in each state serves the purpose of recording, at each point during the computation, where does the name of register i of \mathcal{A} reside in \mathcal{A}' : $f(i) = r$ ($= b, y$) means it is in history i_r (i_b, i_y resp.). If register i is empty then we set $f(i) = \emptyset$. Finally, we include in δ' precisely the following transitions.

- For each $q \xrightarrow{i, j} q' \in \delta$, add $(q, f) \xrightarrow{i_r, j_\phi} (q', f[i \mapsto \emptyset][j \mapsto \phi])$ where $f(i) = r$, $\phi \neq \emptyset$ and $j \neq i \implies f(j) \neq r$.
- For each $q \xrightarrow{i, \emptyset} q' \in \delta$, add $(q, f) \xrightarrow{i_r, \emptyset} (q', f[i \mapsto \emptyset])$ where $f(i) = r$.
- For each $q \xrightarrow{\emptyset, j} q' \in \delta$, add $(q, f) \xrightarrow{i_{\phi'}, j_\phi} (q', f[j \mapsto \phi])$ where $f(j) \neq r$ and $\phi' \in \{b, y\} \setminus \{f(i)\}$.
- For each $q \xrightarrow{\emptyset, \emptyset} q' \in \delta$, add $(q, f) \xrightarrow{i_\phi, \emptyset} (q', f)$ where $\phi' \in \{b, y\} \setminus \{f(i)\}$.
- For each $q \xrightarrow{\emptyset, j} q' \in \delta$, add $(q, f) \xrightarrow{\emptyset, j_\phi} (q', f[j \mapsto \phi])$ where $f(j) \neq r$.
- For each $q \xrightarrow{\emptyset, \emptyset} q' \in \delta$, add $(q, f) \xrightarrow{\emptyset, \emptyset} (q', f)$.

Note that the conditions on f and outgoing labels always ensure that histories i_r contain at most one name. We claim that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Let first $w \in \mathcal{L}(\mathcal{A}')$ have an accepting transition path p in \mathcal{A}' with edges $(q_k, f_k) \xrightarrow{X_k, Y_k} (q_{k+1}, f_{k+1})$. Reading the definition of δ' backwards, this yields an accepting transition path p' in \mathcal{A} with edges $q_k \xrightarrow{X'_k, Y'_k} q_{k+1}$ where

- $X'_k = i$ if $X_k = i_r$, and $X_k = \emptyset$ otherwise;
- $Y'_k = j$ if $Y_k \in \{j_r, j_b, j_y\}$, and $Y_k = \emptyset$ otherwise.

To see that p' accepts w , note that, for each configuration (q_k, f_k, H_k) ,

- if $X_k = i_r$ then $H_k(i_r) = \{a\}$ and $f_k(i) = r$ so a is the last name stored in either of i_r, i_b, i_y ;
- if $X_k = i_b$ then $f_k(i) \neq b$ and therefore the last name a stored in either of i_r, i_b, i_y is not stored in i_b , hence a is locally fresh; similarly if $X_k = y$;
- if $X_k = \emptyset$ then the accepted name is globally fresh.

Hence, noting also that by design the last name stored in either of i_r, i_b, i_y (in the run of p) is the same as the name stored in register i (in p'), we have that

$w \in \mathcal{L}(\mathcal{A}')$.

Conversely, let $w = a_1 \cdots a_N \in \mathcal{L}(\mathcal{A})$ have an accepting transition path p with edges $q_k \xrightarrow{X_k, Y_k} q_{k+1}$ ($k = 0, \dots, N-1$). We construct an accepting path p' in \mathcal{A}' with edges $(q_k, f_k) \xrightarrow{X'_k, Y'_k} (q_{k+1}, f_{k+1})$ as follows. We have that $f_0 = \{(i, \emptyset) \mid i \in [m]\}$. Moreover:

- (a) For each position k such that in all previous appearances of a_k in s , say as $a_k = a_{k'}$ with $k' < k$ (if any), $a_{k'}$ is not stored (i.e. $Y_k = \emptyset$), we set $X_k = \emptyset$.
- (b) For each position k such that $Y_k = i$ and the next appearance of a_k in s is some $a_{k'}$ with $X_{k'} = i$, we set $Y_k = i_r$.
- (c) For each position k such that $Y_k = i$ and the next appearance of a_k in s is some $a_{k'}$ with $X_{k'} = \emptyset$, we choose some $Y_k \in \{i_b, i_y\}$.

The rest of the labels in p' and the form of the f_k 's is derived from the above according to the definition of δ' . To show that p' indeed accepts w we need to show that our above labelling is correct and, in particular, that there is a valid choice of labels from $\{i_b, i_y\}$ in the step (c) above. By a valid choice we mean one such that if, for instance, $Y_k = i_b$ then $f_{k'}(i) \neq b$ and therefore $X_{k'}$ can correctly pick a_k . By definition, the value of $f_{k'}(i)$ is determined by the rightmost position l , with $k < l < k'$, which assigns a name in either of i_r, i_b, i_y (since a_k becomes locally fresh from state q_k to $q_{k'}$, such an l always exists). Our constraint, therefore, (in this case) is that $Y_l \neq b$. In particular, if position l falls under case (c) above, we need to choose $Y_l = y$. Thus, it suffices to colour all our positions which fall under case (c) with colours from b, y , such that no inter-related q_k and q_l have the same colour. We claim that such a colouring is always possible. Indeed, we can build a graph \mathcal{G} as follows. The nodes of \mathcal{G} are the positions which fall under case (c), arranged on a line in ascending order, from left to right. Now, for each inter-related positions k and l as above, we add a directed edge from node k to node l . Then, a valid colouring of our positions is possible iff \mathcal{G} is 2-colourable, i.e. if it contains no cycles. Now note that each node in \mathcal{G} has at most one outgoing edge, and all edges go from left to right (above, $k < l$). Hence, \mathcal{G} is 2-colourable and we can therefore build a valid path p' in \mathcal{A}' accepting w . \square

References

1. S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proc. of LICS*, pp. 150–159, 2004.
2. T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theor. Comput. Sci.*, 3(1):85–104, 1977.
3. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-Bounded Analysis For Concurrent Programs With Dynamic Creation of Threads. *Log. Meth. Comput. Sci.*, 7(4), 2011.
4. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. TLCA*, pp. 86–101, 2005.

5. H. Björklund and T. Schwentick. On notions of regularity for data languages *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.
6. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-Variable Logic on Words with Data. In *LICS*, pp. 7–16, 2006.
7. A. Bouajjani, S. Fratani, and S. Qadeer. Context-bounded analysis of multi-threaded programs with dynamic linked structures. In *Proc. CAV*, pp. 207–220, 2007.
8. G. Ciardo. Petri Nets with Marking-Dependent Arc Cardinality. In *Properties and Analysis, Advances in Petri Nets*, pp.179–198, 1994.
9. C. Dufourd, A. Finkel and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. ICALP*, pp. 103–115, 1998.
10. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermannian and Primitive-Recursive Bounds with Dickson’s Lemma. In *Proc. LICS*, pp. 269–278, 2011.
11. A. Finkel and A. Sangnier. Mixing Coverability and Reachability to Analyze VASS with One Zero-Test. In *Proc. SOFSEM*, pp. 394–406, 2010.
12. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
13. R. Grigore, D. Distefano, R. L. Petersen and N. Tzevelekos. Runtime Verification Based on Register Automata. Submitted.
14. P. Jancar. Undecidability of Bisimilarity for Petri Nets and Some Related Problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.
15. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *Proc. LICS*, pp. 56–66, 1999.
16. A. Kaiser, D. Kroening and T. Wahl. Efficient coverability analysis by proof minimization. In *Proc. CONCUR*, 2012. <http://www.cprover.org/bfc/>
17. M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
18. J. Laird. A fully abstract trace semantics for general references. In *Proc. ICALP ’07*, pp. 667–679, 2007.
19. R. Lipton. The Reachability Problem Requires Exponential Space. Technical Report 62, Yale University, 1976.
20. A. Manuel and R. Ramanujam. Class Counting Automata on Datawords. *Int. J. Found. Comput. Sci.*, 22(4):863–882, 2011.
21. U. Montanari and M. Pistore. An introduction to History Dependent Automata. *Electr. Notes Theor. Comput. Sci.*, 10, 1997.
22. A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *Proc. ESOP*, pp. 419–438, 2011.
23. A. S. Murawski and N. Tzevelekos. Algorithmic Games for Full Ground References. In *Proc. ICALP*, pp. 312–324, 2012.
24. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
25. A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What’s new? In *Proc. MFCS*, pp. 122–141, 1993.
26. C. Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 6(2):223–231, *ESSLLI*, 1978.
27. Ph. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. MFCS*, pp. 616–628, 2010.
28. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proc. CSL ’06*, pp. 41–57, 2006.

29. I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.
30. N. Tzevelekos. Fresh-Register Automata. In *POPL*, pp. 295–306, 2011.